SAPIENZA
UNIVERSITÀ DI ROMA

Report

Progetto di Sistemi Operativi

Progetto scelto

Gioco multiplayer

Studenti

Andrea Misuraca - 1709259
Michele Anselmi - 1716741

Premessa

Il progetto scelto pone come obiettivo lo sviluppo di un videogioco multiplayer, implementato attraverso un server, nel quale vengono sviluppate tutte le dinamiche del gioco, e un client in cui avviene la rappresentazione grafica dello stesso.

Il Progetto

Il gioco in sé per sé è un classico open world da esplorare in terza persona a bordo di un veicolo, con un motivo a scelta del giocatore da dichiarare come secondo argomento all'avvio da Command Line, in cui puoi incontrare altri veicoli appartenenti ad altri giocatori connessi allo stesso server.

Inizializzazione

Il primo passo per far avviare il gioco è runnare il server che accetterà eventuali connessioni di giocatori:

./so_game_server <elevation map> <texture map>

Il primo argomento è una immagine in scala di grigi che definisce, in base all'intensità del grigio, l'altezza delle varie parti della mappa, mentre il secondo argomento è l'immagine da modellare e che, mixata con le altezze dettate dalla *elevation map*, formerà la rappresentazione grafica del mondo.

Dopo aver avviato il server possiamo partecipare al gioco semplicemente scrivendo da Command Line:

./so_game_client <Indirizzo IP server> <texture veicolo>

Il programma caricherà in memoria la texture da caricare sul veicolo e la invierà al server, dopodiché il gioco verrà aperto come finestra detached sullo schermo dell'utente.

Fase 1

Server e setup iniziale

Il server prende come due argomenti da Command Line, come abbiamo già detto, l'elevation map e la texture map, che costituiscono come viene rappresentato il mondo, caricandole in memoria in variabili *Image* surface_elevation* e *Image* surface_texture.*

Successivamente vengono aperte le socket TCP e la socket UDP, attraverso le porte descritte rispettivamente dalla *#define TCP_PORT* e dalla *#define UDP_PORT*.

Viene allocata in memoria una variabile *users* che servirà a mantenere la lista dei giocatori connessi al gioco e viene creato il mondo da rappresentare attraverso la funzione *World_init*.
Quando il setup principale è pronto, vengono lanciati 3 diversi thread :

1. ***TCP_connection***

   gestisce le connessioni di nuovi utenti e le disconnessioni di quelli già connessi

2. ***UDP_sender_handler***

   Invia ad intervalli regolari un pacchetto UDP *ClientUpdate* con tutti gli aggiornamenti sui giocatori, a tutti i giocatori

3. ***UDP_receiver_handler***

   Riceve aggiornamenti tramite messaggi UDP dai giocatori delle rispettive forze di rotazione e traslazione, che vengono elaborati attraverso la funzione *World_update* dalla quale si potranno ottenere *x, y, theta* di ogni giocatore.

Fase 2

Connessione client e handshake iniziale

Il client carica come argomento l'indirizzo IP di riferimento del server e la texture da renderizzare sul veicolo.

Dopo aver caricato proprio la texture in una variabile *Image* my_texture* , viene aperta una socket TCP all'indirizzo specificato nel primo argomento.

Utilizzando questa socket, andremo a richiedere al server i vari elementi che costituiscono il gioco, attraverso la funzione *serverHandshake.*

ServerHandshake prende come argomento il descrittore alla socket TCP, aperta in precedenza, e gli indirizzi a memoria dell'ID del giocatore, della texture del veicolo, della map elevation, della map texture e di una area di memoria dedicata alla texture del veicolo, ma inviata dal server. Quest'ultima sarà quella renderizzata, così da rendere omogenea la rappresentazione della texture in tutti i client connessi al server.

Il server, attraverso il *type* contenuto nel *PacketHeader* ricevuto, riesce a capire come deserializzare il pacchetto, utilizzando le già presenti funzioni di *packet_serialize / deserialize*, e rispondere al client in base alla richiesta.

Ricevute queste informazioni, possiamo inizializzare e renderizzare il mondo vero e proprio attraverso le funzioni *World_init* e *WorldViewer_runGlobal*, per poter poi aggiungere successivamente il proprio veicolo al gioco prima inizializzandolo con la *Vehicle_init* e dopo inserendolo con la *World_addVehicle*.

Setup iniziale terminato, il main process lancia 4 thread secondari, che serviranno a ricevere / inviare / visualizzare gli aggiornamenti:

1. TCP_connection

Riceve messaggi TCP dal server sulla connessione e disconnessione degli utenti

2. UDP_Sender

Invia messaggi UDP al server contenenti informazioni sul proprio veicolo

3. UDP_Receiver

Riceve messaggi UDP dal server contenenti gli aggiornamenti sull'attuale stato del mondo

4. Runner_thread

Renderizza, a intervalli regolari, i nuovi stati del mondo ricevuti in UDP.

Fase 3

Meccanismo di aggiornamento

Una volta lanciati i tutti i thread secondari, l'utente può giocare tranquillamente al gioco utilizzando le frecce direzionali per muoversi intorno al mondo.
I valori delle forze rotazionali e traslazionali della macchinetta vengono inviati al server attraverso il ***UDP_Sender*** , che riceve questi dati e li elabora.

Scattato il timer dettato dalla costante *TIME_TO_SLEEP*, il server prende tutti gli aggiornamenti ricevuti tramite UDP e li manda in broadcast a tutti gli utenti tramite il thread ***UDP_sender***.

Questi pacchetti UDP vengono accolti dal client attraverso la ***UDP_Receiver***, con la quale rimane in attesa di nuovi pacchetti UDP dal server attraverso la funzione recvfrom, e vengono deserializzati, per poter aggiornare con i nuovi valori ricevuti dal server, i veicoli renderizzati nel proprio mondo.

Nel frattempo, seguendo il metronomo dettato dalla variabile *TIME_TO_SLEEP*, il client thread ***updater_thread*** aggiorna la rappresentazione del mondo lanciando la funzione *World_update*, per poi rimettersi nello stato di sleep.

Fase 4

Gestione nuove connessioni / disconnessioni

Il server rimane in ascolto, attraverso la socket TCP, di nuove richieste di connessione da parte di giocatori che si vogliono aggiungere al gioco.
Quando viene accettata una richiesta di partecipazione e dopo aver ottenuto sia ID utente e sia la texture del veicolo, il server gira in broadcast TCP un pacchetto *ImagePacket* contenente il *type = UserConnected, id =* codice ID del nuovo utente connesso, *Image =* texture da renderizzare sul veicolo del nuovo utente.

Per quanto riguarda la disconnessione dei giocatori, ci serviamo dalla chiusura della socket TCP con il server, la quale triggera l'eliminazione del veicolo dell'utente nel mondo e l'invio in broadcast TCP di un pacchetto *IdPacket* contentente il *type = UserDisconnected* e l'*id = codice dell'utente disconnesso.*

Questi messaggi TCP vengono presi dal client attraverso il thread ***TCP_connections_receiver*** e gestiti di conseguenza con l'inserimento o la rimozione dell'utente nel mondo.

Fase 5

Gestione segnali & cleanup

Entrambi i moduli, client e server, hanno implementato la gestione dei segnali di interrupt che si possono verificare nel corso dell'esecuzione.

La ricezione di ***SIGHUP, SIGINT*** e ***SIGTERM*** triggererà il richiamo della funzione *cleanMemory(),* la quale si preoccupa di interrompere tutti i thread in esecuzione, settando una variabile *running = 0*, di pulire la memoria dai thread attraverso la *pthread_cancel*, di chiudere le socket TCP e UDP, deallocare il mondo "distruggendolo" utilizzando la funzione *World_destroy* e di liberare lo spazio allocato dalle immagini della *Elevation Map,* della *Texture Map* e del veicolo attraverso la *Image_free.*